

Assignment-3

1. (a) **False.**

Because \mathbf{A} is nonsingular, there are a number of properties that are associated with it. One of these is the fact that \mathbf{A} has a unique solution for all \mathbf{b} . Likewise if $\mathbf{b} = \mathbf{0}$, then \mathbf{x} is necessarily the 0 vector.

This question was done assuming that $\mathbf{A} = n \times n \rightarrow \mathbf{b} = 1 \times n$. If \mathbf{b} is of size $m \neq n$, then $\mathbf{Ax} = \mathbf{b}$ has either zero or infinitely many solutions. However, this does not depend on the contents of \mathbf{b} , but rather the initial parameters of the problem.

(b) **False.**

This can be proven false simply by taking the determinant of a 3×3 matrix.

$$\det(\mathbf{A}) = a_{11}(a_{22}a_{33} + a_{23}a_{32}) - a_{12}(a_{21}a_{33} + a_{23}a_{31}) + a_{13}(a_{21}a_{32} + a_{22}a_{31})$$

As there are terms in this equation that are not dependant on a_{ii} , a matrix with a zero on the principle diagonal is not necessarily singular.

(c) **False.**

The conditioning of a matrix \mathbf{A} is related to the largest sum of a single column (the 1-norm). The partial pivoting algorithm scans the rest of the column below the current pivot point (a_{kk}) for the largest value.

A well-conditioned matrix would simply have small column sums, which does not necessarily affect the location of the largest values in each row, which in turn does not affect the need for partial pivoting.

(d) **True.**

By definition, the condition number of a matrix $\mathbf{A} = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$. Therefore, the condition number of $\mathbf{A}^{-1} = \|\mathbf{A}^{-1}\| \|(\mathbf{A}^{-1})^{-1}\| = \|\mathbf{A}^{-1}\| \|\mathbf{A}\|$. As $\|\mathbf{A}\|$, $\|\mathbf{A}^{-1}\|$ are scalar values, the two products are the same.

(e) **False.**

If all lines are parallel, there are zero intersections, and therefore zero solutions. If all lines are the same, there are infinitely many intersections, and therefore infinitely many solutions. If all lines intersect at exactly one point, there is exactly one solution. There are no other possibilities, as two linear equations cannot intersect at 2 points unless they intersect at infinitely many points.

(f) One of the reasons pivoting is essential in the G.E. method of solving matrices is to avoid potential zeroes on the principle diagonal. When the G.E. is being done, each element on the principle diagonal determines the multiplier for the rows below, $m_{ij} = \frac{a_{ij}}{a_{jj}}$. Since

the diagonal element is in the denominator, it must be nonzero. This is a mathematical reason for pivoting.

The other reason is a numerical one. If the current pivot point is a_{jj} , and this value is very small (10^{-16}), then the multiplier that results will be extremely large. If the values in column j are then relatively small to a_{jj} , the subtraction will result in a loss of significance, due to not being able to express numbers such as $10^{16} - 1$ accurately. Partial pivoting moves the row with the largest j^{th} element to the current row, then computes the multiplier. This avoids the large multiplier result, and will aid in maintaining significance.

- (g) The condition number is a nontrivial computation due to the computation of the norms that make up the number. In order to compute the 1-norm, which is common, the algorithm must sum every column, the largest of which becomes the condition number. This alone requires $O(n^2)$ operations. The algorithm must then compute the inverse of \mathbf{A} , which is an even more expensive operation. More curcially, \mathbf{A}^{-1} must be completely recomputed if \mathbf{A} is even slightly modified.
- (h) On its own, the relative residual does not carry much weight. A small residual would, at first glance, seem to imply low error, and therefore higher accuracy. However, this is not the case, and it turns out that the condition number of a matrix is directly involved in the accuracy of the solution. In order for a solution to be accurate, the solution should not only have a small relative residual, but should also be moderately well conditioned.
- (i)
 - i. $\det(\mathbf{cA}) = c^n * \det(\mathbf{A})$
 In this case, \mathbf{A} would be an $n \times n$ diagonal matrix, with every diagonal entry = 1, and $c = \frac{1}{2}$. The determinant of a diagonal matrix with every diagonal value = 1 is simply the identity matrix, and $\det(\mathbf{I})$ is 1. Therefore, $\det(\frac{1}{2}\mathbf{I}) = (\frac{1}{2})^n$.
 - ii. Assume the use of the 1-norm
 $\|\mathbf{A}\|_1 = \frac{1}{2}$
 Because the inverse of a diagonal matrix is found by simply inverting all diagonal elements, $\|\mathbf{A}^{-1}\|_1 = 2$
 $K(\mathbf{A}) = \|\mathbf{A}\|_1 * \|\mathbf{A}^{-1}\|_1 = 1$
 - iii. Any scalar multiple of the identity matrix will not amplify input errors. This makes intuitive sense, as the condition number of a matrix sets an upper bound on the maximum distortion of the result due to input errors. Because each element in the system (x_1, x_2, \dots, x_n) only appears once in \mathbf{A} , each entry simply takes the corresponding \mathbf{b} value ($a_1 = b_1, a_2 = b_2, \dots, a_n = b_n$). Any error that is present in the input will be directly translated to the output.

2. (a)

$$\begin{bmatrix} 1 & 2 \\ 2 & 4.01 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 6.01 \end{bmatrix}$$

$$x_1 = 3 - 2x_2$$

$$2(3 - 2x_2) + 4.01x_2 = 6.01$$

$$6 + 0.01x_2 = 6.01$$

$$0.01x_2 = 0.01$$

$$x_2 = 1$$

$$x_1 = 1$$

(b)

$$\begin{bmatrix} 1 & 2 \\ 2 & 4.01 \end{bmatrix} \begin{bmatrix} -600 \\ 301 \end{bmatrix} = \begin{bmatrix} 2 \\ 7.1 \end{bmatrix}$$

Here,

$$x^* = \begin{bmatrix} -600 \\ 301 \end{bmatrix}$$

and

$$b^* = \begin{bmatrix} 2 \\ 7.1 \end{bmatrix}$$

The relative forward error is defined by $\frac{\|x^*\|_\infty}{\|x\|_\infty}$, while the relative backward error is $\frac{\|b^*\|_\infty}{\|b\|_\infty}$.

Relative Forward Error	Relative Backward Error
$\ x^*\ = 600$	$\ b^*\ = 7.1$
$\ x\ = 1$	$\ b\ = 6.01$
$\frac{600}{1} = 600$	$\frac{7.1}{6.01} \approx 1.18 \dots$

The error magnification factor is (relative forwards error \div relative backwards error).

$$E_m = 600 \div \frac{7.1}{6.01} \approx 507.89 \dots$$

(c) $K(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$

$$\|\mathbf{A}\| = 6.01$$

$$\mathbf{A}^{-1} = \frac{1}{0.01} \begin{bmatrix} 4.01 & -2 \\ -2 & 1 \end{bmatrix}$$

$$\|\mathbf{A}^{-1}\| = 201$$

$$K(\mathbf{A}) = 201 * 6.01 = 1208.01$$

The poor conditioning of the matrix is a clear indication that a small change to the data will cause a large change in the result. In this case, the altered data that gave rise to the vector x^* was b^* . The large condition number sets the upper bound on the relative output error, meaning that x^* can grow extremely large, without producing extremely different results in b^* .

3. (a) Begin with the augmented matrix $\mathbf{A}|\mathbf{b}$

$$\mathbf{A}|\mathbf{b} = \left[\begin{array}{ccccc} 1 & 2 & 3 & 4 & -6 \\ 2 & 3 & 7 & 9 & -17 \\ -2 & -6 & -2 & -4 & -4 \\ 1 & 0 & 11 & 15 & -34 \end{array} \right]$$

Step 1: Pivot point a_{11}

First, we must employ partial pivoting to ensure there are no multipliers > 1 . We see that row 2 has the largest value in column 1, so we use this as the first row instead. The permutation matrix associated with swapping rows $R1$ and $R2$ is

$$\mathbf{P} = \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array} \right]$$

and the matrix we are now operating on is

$$\mathbf{A} \left[\begin{array}{ccccc} 2 & 3 & 7 & 9 & -17 \\ 1 & 2 & 3 & 4 & -6 \\ -2 & -6 & -2 & -4 & -4 \\ 1 & 0 & 11 & 15 & -34 \end{array} \right]$$

We now calculate the multipliers needed to zero out the first column.

$$m_{21} = \frac{a_{21}}{a_{11}} = 0.5 \quad \parallel \quad m_{31} = \frac{a_{31}}{a_{11}} = -1 \quad \parallel \quad m_{41} = \frac{a_{41}}{a_{11}} = 0.5$$

Then, carry out the row operations on $R2, R3, R4$.

$$R2 \rightarrow R2 - m_{21}R1 \quad \parallel \quad R3 \rightarrow R3 - m_{31}R1 \quad \parallel \quad R4 \rightarrow R4 - m_{41}R1$$

The new matrix generated after these operations is

$$\mathbf{A} = \left[\begin{array}{ccccc} 2 & 3 & 7 & 9 & -17 \\ (0.5) & 0.5 & -0.5 & -0.5 & -2.5 \\ (-1) & -3 & -5 & 5 & -21 \\ (0.5) & -1.5 & 7.5 & 10.5 & -25.5 \end{array} \right]$$

where the normal zeros are replaced by the multipliers.

Step 2: Pivot point a_{22}

Again, start by swapping rows according to partial pivoting. This includes changing the positions of the multipliers, although their values will not be affected by the row operations that follow. It also includes swapping the corresponding rows in the permutation matrix. In this case, we swap rows $R2$ and $R3$.

$$\mathbf{A} = \left[\begin{array}{ccccc} 2 & 3 & 7 & 9 & -17 \\ (-1) & -3 & -5 & 5 & -21 \\ (0.5) & 0.5 & -0.5 & -0.5 & -2.5 \\ (0.5) & -1.5 & 7.5 & 10.5 & -25.5 \end{array} \right]$$

and

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next, calculate the new multipliers. In this step there are only two, as the only rows that have items to be zeroed out are $R3$ and $R4$.

$$m_{32} = \frac{a_{32}}{a_{22}} = -\frac{1}{6} \parallel m_{42} = \frac{a_{42}}{a_{22}} = 0.5$$

Next, perform the row operations on $R3, R4$.

$$R3 \rightarrow R3 - m_{32}R2 \parallel R4 \rightarrow R4 - m_{42}R2$$

The new matrix is

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 7 & 9 & -17 \\ (-1) & -3 & -5 & 5 & -21 \\ (0.5) & (-\frac{1}{6}) & \frac{1}{3} & \frac{1}{3} & -6 \\ (0.5) & (0.5) & 5 & 8 & 36 \end{bmatrix}$$

again, with the normal zeros replaced by the multipliers.

Step 3: Pivot point a_{33}

In this case, we have to switch $R3$ and $R4$ in \mathbf{A} and \mathbf{P} according to partial pivoting.

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 7 & 9 & -17 \\ (-1) & -3 & -5 & 5 & -21 \\ (0.5) & (0.5) & 5 & 8 & 36 \\ (0.5) & (-\frac{1}{6}) & \frac{1}{3} & \frac{1}{3} & -6 \end{bmatrix}$$

and

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The only multiplier to be calculated is m_{43}

$$m_{43} = \frac{a_{43}}{a_{33}} = \frac{1}{15}$$

and the only row operation to be performed is

$$R4 \rightarrow R4 - m_{43}R3$$

The final matrix is

$$\mathbf{A} = \begin{bmatrix} 2 & 3 & 7 & 9 & -17 \\ (-1) & -3 & -5 & 5 & -21 \\ (0.5) & (0.5) & 5 & 8 & 36 \\ (0.5) & (-\frac{1}{6}) & (\frac{1}{15}) & -0.2 & -8.4 \end{bmatrix}$$

From here, we can extract both the lower and upper triangular matrices \mathbf{L} , \mathbf{U} .

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0.5 & 0.5 & 1 & 0 \\ 0.5 & -\frac{1}{6} & \frac{1}{15} & 1 \end{bmatrix} \quad \mathbf{U} = \begin{bmatrix} 2 & 3 & 7 & 9 \\ 0 & -3 & 5 & 5 \\ 0 & 0 & 5 & 8 \\ 0 & 0 & 0 & -0.2 \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

- (b) When checked in MATLAB, the results of the hand calculation agree with the computed result.

`lup_check.m` script used to generate matrices.

```
% generate LU decomposition of matrix A

A = [1 2 3 4; 2 3 7 9; -2 -6 -2 -4; 1 0 11 15];
[L,U,P] = lu(A);

%Display results
fprintf('L =\n');
disp(L);

fprintf('\nU =\n');
disp(U);

fprintf('\nP =\n');
disp(P);
```

Output of `lup_check.m`

```
L =
    1.0000         0         0         0
   -1.0000    1.0000         0         0
    0.5000    0.5000    1.0000         0
    0.5000   -0.1667    0.0667    1.0000

U =
    2.0000    3.0000    7.0000    9.0000
         0   -3.0000    5.0000    5.0000
         0         0    5.0000    8.0000
         0         0         0   -0.2000

P =
     0     1     0     0
     0     0     1     0
     0     0     0     1
     1     0     0     0
```

(c) If the decomposition is correct, $\mathbf{LU} == \mathbf{AP}$

$$\begin{aligned}\mathbf{LU} &= \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} & L_{11}U_{13} & L_{11}U_{14} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} & L_{21}U_{13} + L_{22}U_{23} & L_{21}U_{14} + L_{22}U_{24} \\ L_{31}U_{11} & L_{31}U_{12} + L_{32}U_{22} & L_{31}U_{13} + L_{32}U_{23} + L_{33}U_{33} & L_{31}U_{14} + L_{32}U_{24} + L_{33}U_{34} \\ L_{41}U_{11} & L_{41}U_{12} + L_{42}U_{22} & L_{41}U_{13} + L_{42}U_{23} + L_{43}U_{33} & L_{41}U_{14} + L_{42}U_{24} + L_{43}U_{34} + L_{44}U_{44} \end{bmatrix} \\ &= \begin{bmatrix} 2 & 3 & 7 & 9 \\ -2 & (-3-3) & (-7+5) & (-9+5) \\ 1 & (1.5-1.5) & (3.5+2.5+5) & (4.5+2.5+8) \\ 1 & (1.5+0.5) & (3.5-\frac{5}{6}+\frac{1}{3}) & (4.5-\frac{5}{6}+\frac{8}{15}-0.2) \end{bmatrix} \\ &= \begin{bmatrix} 2 & 3 & 7 & 9 \\ -2 & -6 & -2 & -4 \\ 1 & 0 & 11 & 15 \\ 1 & 2 & 3 & 4 \end{bmatrix}\end{aligned}$$

The product \mathbf{AP} can be found analytically by taking the row operations needed to convert the identity matrix \mathbf{I} into \mathbf{P} , and applying them to \mathbf{A} .

$$\mathbf{P} = (R1_I \leftrightarrow R2_I) \rightarrow (R2_I \leftrightarrow R3_I) \rightarrow (R3_I \leftrightarrow R4_I)$$

Performing the same operations on \mathbf{A} yields

$$\mathbf{AP} = \begin{bmatrix} 2 & 3 & 7 & 9 \\ -2 & -6 & -2 & -4 \\ 1 & 0 & 11 & 15 \\ 1 & 2 & 3 & 4 \end{bmatrix}$$

The two matrices are equal, and there is no residual.

4. The matrix \mathbf{A} is symmetric positive definite if the determinants of all leading principle submatrices are positive.

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 10 \end{bmatrix}$$

Here, the determinant of the first principle submatrix is simply 4, the second is $((4 * 2) - (-2 * -2)) = 4$, and the third is $4((2 * 10) - (-3 * -3)) + 2((-2 * 10) - 0) + 0 = 4$.

The Cholesky decomposition says that $\mathbf{LL}^T = \mathbf{A}$, meaning

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Step 1: Multiply $R1$ of \mathbf{L} with columns of \mathbf{L}^T . This result is the first row of \mathbf{A} .

$$\begin{aligned} (l_{11})^2 &= 4 & l_{11} &= 2 \\ l_{11}l_{21} &= -2 & l_{21} &= -1 \\ l_{11}l_{31} &= 0 & l_{31} &= 0 \end{aligned}$$

Step 2: Multiply $R2$ of \mathbf{L} with columns of \mathbf{L}^T . This result is the second row of \mathbf{A} . Since the first equation ($l_{21}l_{11}$) does not have any unknowns, we can skip this calculation.

$$\begin{aligned} (l_{21})^2 + (l_{22})^2 &= 2 & l_{22} &= 1 \\ l_{21}l_{31} + l_{22}l_{32} &= -3 & l_{32} &= -3 \end{aligned}$$

Step 3: Multiply $R3$ of \mathbf{L} with columns of \mathbf{L}^T . This result is the second row of \mathbf{A} . Since The first equation ($l_{32}l_{11}$), and the second equation ($l_{31}l_{21} + l_{32}l_{22}$) do not have any unknowns, we can skip them, and are left with a single calculation for l_{33} .

$$(l_{31})^2 + (l_{32})^2 + (l_{33})^2 = 10 \quad l_{33} = 1$$

$$\mathbf{L} = \begin{bmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -3 & 1 \end{bmatrix}$$

Just to make sure that this is a valid Cholesky decomposition,

$$\begin{aligned} \begin{bmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -3 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix} &= \begin{bmatrix} (2*2) & (2*-1) & 0 \\ (2*-1) & (-1*-1) + (1*1) & (-3*1) \\ 0 & (-3*1) & (-3*-3) + (1*1) \end{bmatrix} \\ &= \begin{bmatrix} 4 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 10 \end{bmatrix} \end{aligned}$$

5. (a) See below

`myA.m` function

```
% myA.m
%
% Fill out an nxn matrix, where each entry a_{ij} is determined by the
% formula a_{ij} = 5/(i+2j-1). Return the result b=Ax, where x is a size
% n vector of ones.

function [A,b] = myA(n)
% input:
% n = size of square matrix A
%
% output:
```



```

% A = resulting matrix A
% b = Ax

% initialize A as a zero matrix
A = zeros(n);

% loops to populate A
for i = 1:n
    for j = 1:n
        A(i,j) = 5/(i+2*j-1);
    end
end

% create x vector
x = ones(n,1);

% calculate b vector
b = A*x;

```

- (b) The results of `myA.m` for size $n = 5 : 20$ are generated by a second script, `runmyA.m`.

To summarize, the condition number of \mathbf{A} ($\kappa(\mathbf{A})$) grows increasingly large with n . This causes the relative error to increase as well, causing the significance of the result to deteriorate, until there is no significance left at $n = 17$.

Note: The `fprintf` command that displays the results is too long to be displayed on a single line. The ending variables that are used are `nE`, `kA`, `rr`, `scaledR`, which are the norm of the error, the condition number of \mathbf{A} , the relative residual, and the upper bound on the error, respectively.

`runmyA.m`

```

% Script to run myA.m over values of n = 5:20, or until the error grows
% large enough that there is no significance in the results
clc;

%supress warnings
warning('off','all');

for n = 5:20
    % get calculated values of the solution vector x
    [A,b] = myA(n);
    xc = A\b;

    % define correct solution, x
    x = ones(n,1);

    % calculate residual
    r = b - A*xc;
    % then the infinity norm of the residual

```

```

nr = norm(r, Inf);

% calculate error
E = xc - x;
% then the infinity norm of the error
nE = norm(E, Inf);

% estimate condition number of A
% (this should grow with increasing n)
kA = cond(A, Inf);

% calculate relative error
rE = nE / norm(xc, Inf);

% calculate relative residual
rr = nr / (norm(A, Inf)*norm(xc, Inf));

% values values for error bound inequality
scaledR = kA * rr;

% display results
fprintf('\nn = %d\n', n);
fprintf('computed solution\n');
disp(xc);
fprintf('residual norm    error norm        K(A)                relative residual\n');
upper error bound\n');
fprintf('%10.5e    %10.5e    %10.5e    %10.5e    %10.5e    %10.5e\n', nr, nE, rr, scaledR);
fprintf('\nrelative error\n');
fprintf('%10.5e\n', rE);

% stop the loop if re > 1
if rE >= 1
    fprintf('Error has exceeded 100%%.\n');
    return;
end

end

```

Output of runmyA.m script

```

n = 5
computed solution
1.0000
1.0000
1.0000
1.0000
1.0000

residual norm    error norm        K(A)                relative residual    upper error bound
8.88178e-16      2.75980e-11      1.97907e+06      1.55593e-16      3.07930e-10

```

relative error
2.75980e-11

n = 6
computed solution
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000

residual norm	error norm	K(A)	relative residual	upper error bound
8.88178e-16	8.66516e-11	7.03420e+07	1.45009e-16	1.02002e-08
relative error				
8.66516e-11				

n = 7
computed solution
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000

residual norm	error norm	K(A)	relative residual	upper error bound
8.88178e-16	2.57584e-08	2.44925e+09	1.37019e-16	3.35594e-07
relative error				
2.57584e-08				

n = 8
computed solution
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000
1.0000

residual norm	error norm	K(A)	relative residual	upper error bound
4.44089e-16	1.39412e-07	9.72859e+10	6.53587e-17	6.35848e-06
relative error				
1.39412e-07				

n = 9
computed solution
1.0000

1.0000				
1.0000				
1.0000				
1.0000				
1.0000				
1.0000				
1.0000				
1.0000				
1.0000				
residual norm	error norm	K(A)	relative residual	upper error bound
8.88178e-16	3.42030e-05	3.65077e+12	1.25580e-16	4.58464e-04
relative error				
3.42021e-05				
n = 10				
computed solution				
1.0000				
1.0000				
1.0000				
1.0000				
0.9999				
1.0004				
0.9992				
1.0009				
0.9995				
1.0001				
residual norm	error norm	K(A)	relative residual	upper error bound
1.77636e-15	9.10138e-04	1.31367e+14	2.42371e-16	3.18397e-02
relative error				
9.09311e-04				
n = 11				
computed solution				
1.0000				
1.0000				
1.0000				
1.0004				
0.9964				
1.0162				
0.9570				
1.0693				
0.9337				
1.0346				
0.9924				
residual norm	error norm	K(A)	relative residual	upper error bound
8.88178e-16	6.93016e-02	4.80162e+15	1.10020e-16	5.28273e-01
relative error				
6.48101e-02				

n = 12

computed solution

1.0000
1.0000
0.9998
1.0039
0.9601
1.2285
0.2077
2.7222
-1.3593
2.9761
0.0764
1.1844

residual norm	error norm	K(A)	relative residual	upper error bound
1.77636e-15	2.35928e+00	1.91125e+17	7.69365e-17	1.47045e+01
relative error				
7.92745e-01				

n = 13

computed solution

1.0000
1.0000
1.0002
0.9961
1.0461
0.6916
2.2721
-2.3804
6.8928
-5.6958
5.7738
-0.9384
1.3419

residual norm	error norm	K(A)	relative residual	upper error bound
8.88178e-16	6.69576e+00	8.65998e+17	1.62077e-17	1.40358e+01
relative error				
9.71419e-01				

n = 14

computed solution

1.0000
1.0000
1.0007
0.9829
1.1976
-0.2557
5.6692
-9.1271

12.1257
0.8049
-14.8577
21.1347
-10.0368
3.3615

residual norm	error norm	K(A)	relative residual	upper error bound
1.77636e-15	2.01347e+01	3.41375e+18	1.03396e-17	3.52968e+01
relative error				
9.52684e-01				

n = 15

computed solution

1.0000
1.0000
1.0005
0.9891
1.1084
0.4433
2.4592
-0.4353
-0.2028
2.5544
10.9906
-26.7285
31.7774
-15.4747
4.5184

residual norm	error norm	K(A)	relative residual	upper error bound
1.33227e-15	3.07774e+01	1.50673e+18	5.05391e-18	7.61488e+00
relative error				
9.68531e-01				

n = 16

computed solution

1.0000
1.0000
0.9988
1.0307
0.6059
3.7977
-10.6929
29.8327
-36.6665
11.3318
38.5622
-38.3413
-15.5495
54.5971

```

-34.7402
 9.2333

residual norm  error norm  K(A)      relative residual  upper error bound
2.66454e-15    5.35971e+01    3.72142e+18    5.77433e-18      2.14887e+01
relative error
9.81684e-01

n = 17
computed solution
 1.0000
 1.0000
 1.0005
 0.9880
 1.1435
 0.0684
 4.4167
-5.5352
 3.8331
14.7443
-25.8951
 9.6909
31.4856
-45.1613
29.2959
-6.6839
 1.6085

residual norm  error norm  K(A)      relative residual  upper error bound
1.77636e-15    4.61613e+01    2.61016e+18    4.57427e-18      1.19396e+01
relative error
1.02214e+00
Error has exceeded 100%.

```

6. (a) By definition, $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$.

If \mathbf{A} is an $n \times n$ matrix, then so are \mathbf{A}^{-1} and \mathbf{I} . When the multiplication $\mathbf{A}\mathbf{A}^{-1}$ is carried out, there are exactly n terms that result in a 1, and they lie on the principle diagonal.

The rest of the terms are necessarily zero. The expressions that produce the diagonal ones are

$$\begin{aligned}
 &a_{11}a_{11}^{-1} + a_{12}a_{21}^{-1} + \cdots + a_{1n}a_{n1}^{-1} \\
 &a_{21}a_{12}^{-1} + a_{22}a_{22}^{-1} + \cdots + a_{2n}a_{n2}^{-1} \\
 &\vdots \\
 &a_{n1}a_{1n}^{-1} + \cdots + a_{nn}a_{nn}^{-1}
 \end{aligned}$$

When taking the product of \mathbf{A} and any of the columns of \mathbf{A}^{-1} , the result is

$$\begin{bmatrix} a_{11}a_{1i}^{-1} + a_{12}a_{2i}^{-1} + \cdots + a_{1n}a_{ni}^{-1} \\ a_{21}a_{1i}^{-1} + a_{22}a_{2i}^{-1} + \cdots + a_{2n}a_{ni}^{-1} \\ \vdots \\ a_{n1}a_{1i}^{-1} + a_{n2}a_{2i}^{-1} + \cdots + a_{nn}a_{ni}^{-1} \end{bmatrix}$$

where $i = 1 : n$.

We know that the only sum that produces a 1 is of the form $a_{i1}a_{1i}^{-1} + a_{i2}a_{2i}^{-1} + \cdots + a_{in}a_{ni}^{-1}$. Clearly, there is only one of these terms in the product shown above, and the rest of the terms are necessarily zero. Therefore, $\mathbf{A}\mathbf{u}_i = \mathbf{v}_i$.

- (b) The calculations for \mathbf{A}^{-1} were carried out in matlab by the script `itinverse.m`. This script generates a random 5×5 matrix, and finds the column vector \mathbf{u}_i such that $\mathbf{A}^{-1}\mathbf{u}_i = \mathbf{I}_i$, where \mathbf{I}_i is the i^{th} column vector of the identity matrix.

`itinverse.m` script

```
% compute the inverse of 5x5 matrix A using the iterative method
% Aui = vi, where ui is the ith column vector of the inverse, and
% vi is a vector with a 1 in the ith place, and zeros in the rest
clc;

% create random matrix A
A = rand(5);
fprintf('Initial matrix A =\n');
disp(A);

% initialize A^{-1}
Ain = zeros(5,1);

% iterate through all columns
for i = 1:5
    % create resultant vector
    v = zeros(5,1);
    v(i) = 1;

    % calculate each column of A^{-1} (ui)
    ui = A\v;
    %disp(ui);

    % add column to inverse matrix
    Ain = cat(2,Ain,ui);
end

% strip initial zero values of Ain
Ain = Ain(:,2:end);
```



```

fprintf('Calculated A^{-1} =\n');
disp(Ain);

% check that A*A^{-1} = I
ic = A*Ain;

fprintf('Calculated I =\n');
disp(ic);

```

(c) The calculated values of $\mathbf{A}^{-1}, \mathbf{I}$ are displayed below.

Output of `itinverse.m`

```

Initial matrix A =
    0.6837    0.6407    0.7400    0.0862    0.7894
    0.1321    0.3288    0.2348    0.3664    0.3677
    0.7227    0.6538    0.7350    0.3692    0.2060
    0.1104    0.7491    0.9706    0.6850    0.0867
    0.1175    0.5832    0.8669    0.5979    0.7719

Calculated A^{-1} =
   -1.1987   -0.4957    2.7429   -1.8838    0.9413
    3.5340    4.4240   -3.8935    3.7401   -5.1016
   -0.7316   -4.1066    1.1127   -0.4566    2.4583
   -2.6877    0.9870    2.3040   -1.5782    1.8406
    0.4161    0.5807   -0.5103   -0.8036    0.8199

Calculated I =
    1.0000   -0.0000    0.0000   -0.0000   -0.0000
    0.0000    1.0000   -0.0000    0.0000   -0.0000
   -0.0000   -0.0000    1.0000   -0.0000   -0.0000
    0.0000    0.0000   -0.0000    1.0000   -0.0000
   -0.0000   -0.0000    0.0000   -0.0000    1.0000

```

7. The solutions to the Lorenz equations are calculated by two MATLAB files. The first, `lorenz.m`, defines the process for computing the next iterate using the given functions and their partial derivatives. The second, `runlorenz.m`, contains a loop that supplies `lorenz.m` values according to Newton's method. When two successive iterates are close enough together, or if the system evaluated at the current result is close enough to zero, the script stops and displays a solution to the system. Below are `lorenz.m`, `runlorenz.m`, and the output for the three solutions. The particular solution reached depends on the initial guesses provided by the user.

`lorenz.m` function

```

% solve the Lorenz system of equations using Newton's method
function [x1,x2,x3] = lorenz(x0,y0,z0)

```

```

% define given initial conditions
s = 10;
r = 28;
b = 8/3;

% define the three functions in the system
f1 = @(x,y,z) s*(y-x);
f2 = @(x,y,z) r*x-y-x*z;
f3 = @(x,y,z) x*y-b*z;

% define functions' partials at points (x0,y0,z0)
f1x = -s;    f2x = r-z0;    f3x = y0;
f1y = s;      f2y = -1;      f3y = x0;
f1z = 0;      f2z = -x0;     f3z = -b;

% define solution vector
b = [f1(x0,y0,z0); f2(x0,y0,z0); f3(x0,y0,z0)];

% define jacobian
A = [f1x f1y f1z; f2x f2y f2z; f3x f3y f3z];

% solution vector
r = A\b;

% assign values of r to x1,x2,x3
x1 = r(1);
x2 = r(2);
x3 = r(3);

```

runlorenz.m script

```

% run successive loops of the Lorenz system to find solutions
clc;

% define constants
s = 10; r = 28; b = 8/3;

% define functions
f1 = @(x,y,z) s*(y-x);
f2 = @(x,y,z) r*x - y - x*z;
f3 = @(x,y,z) x*y - b*z;

% initial guesses
x0 = 5;
y0 = 5;
z0 = 5;

% iteration counter

```

```

it = 0;

% tolerance in the results
tol = 1e-10;

fprintf('Computing zeros within tolerance of %e\n',tol);
fprintf('Initial guesses: x = %d    y = %d    z = %d\n',x0,y0,z0);

% begin the algorithm
while it <= 20
    it = it + 1;

    % calculate new guesses
    [x,y,z] = lorenz(x0,y0,z0);
    x1 = x0 - x;
    y1 = y0 - y;
    z1 = z0 - z;

    fprintf('\nm = %d\nNew guesses:\n', it);
    fprintf('x = %10.5e    y = %10.5e    z = %10.5e\n',x1,y1,z1);

    % check if two successive iterates are close
    if abs(x0-x1) <= tol && abs(y0-y1) <= tol && abs(z0-z1) <= tol
        % assign values and end
        x0 = x1;    y0 = y1;    z0 = z1;
        fprintf('\nTwo successive iterates are within %d in magnitude. Zero has been f
        break
    end

    % check if the functions are close enough to zero
    if abs(f1(x1,y1,z1)) <= (0+tol) &&...
        abs(f2(x1,y1,z1)) <= (0+tol) &&...
        abs(f3(x1,y1,z1)) <= (0+tol)

        % assign values and end
        x0 = x1;    y0 = y1;    z0 = z1;
        fprintf('\nValues of f1, f2, f3 are sufficiently close to zero. Zero has been
        break
    end

    % assign new values to guesses
    x0 = x1;    y0 = y1;    z0 = z1;
end

fprintf('\nSolutions:\nx = %10.5e    y = %10.5e    z = %10.5e\n',x0,y0,z0);

```

Solution set 1 produced by `runlorenz.m`

Computing zeros within tolerance of 1.000000e-10

```

Initial guesses: x = 1    y = 1    z = 1

n = 1
New guesses:
x = -5.44554e-02    y = -5.44554e-02    z = -4.15842e-01

n = 2
New guesses:
x = -8.23834e-04    y = -8.23834e-04    z = -1.07838e-03

n = 3
New guesses:
x = -3.28948e-08    y = -3.28948e-08    z = -2.54493e-07

n = 4
New guesses:
x = -3.10056e-16    y = -3.10056e-16    z = -4.05775e-16

Values of f1, f2, f3 are sufficiently close to zero. Zero has been found

Solutions:
x = -3.10056e-16    y = -3.10056e-16    z = -4.05775e-16

```

Solution set 2 produced by `runlorenz.m`

```

Computing zeros within tolerance of 1.000000e-10
Initial guesses: x = 5    y = 5    z = 5

n = 1
New guesses:
x = -2.21154e+01    y = -2.21154e+01    z = -9.23077e+01

n = 2
New guesses:
x = -8.14002e+00    y = -8.14002e+00    z = -4.83940e+01

n = 3
New guesses:
x = -7.45823e+00    y = -7.45823e+00    z = 2.06851e+01

n = 4
New guesses:
x = -8.75180e+00    y = -8.75180e+00    z = 2.80953e+01

n = 5
New guesses:
x = -8.49425e+00    y = -8.49425e+00    z = 2.70322e+01

n = 6

```

New guesses:
x = -8.48529e+00 y = -8.48529e+00 z = 2.70000e+01

n = 7
New guesses:
x = -8.48528e+00 y = -8.48528e+00 z = 2.70000e+01

n = 8
New guesses:
x = -8.48528e+00 y = -8.48528e+00 z = 2.70000e+01

Two successive iterates are within 1.000000e-10 in magnitude. Zero has been found

Solutions:
x = -8.48528e+00 y = -8.48528e+00 z = 2.70000e+01

Solution set 3 produced by runlorenz.m

Computing zeros within tolerance of 1.000000e-10
Initial guesses: x = 10 y = 10 z = 10

n = 1
New guesses:
x = 8.18966e+00 y = 8.18966e+00 z = 2.39224e+01

n = 2
New guesses:
x = 8.51023e+00 y = 8.51023e+00 z = 2.71205e+01

n = 3
New guesses:
x = 8.48537e+00 y = 8.48537e+00 z = 2.70004e+01

n = 4
New guesses:
x = 8.48528e+00 y = 8.48528e+00 z = 2.70000e+01

n = 5
New guesses:
x = 8.48528e+00 y = 8.48528e+00 z = 2.70000e+01

Values of f1, f2, f3 are sufficiently close to zero. Zero has been found

Solutions:
x = 8.48528e+00 y = 8.48528e+00 z = 2.70000e+01